Lecture 3 - Game Engine Architecture

"If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization." - Gerald Weinberg

- Mapping a game engine architecture against a game architecture
- What subsystems exist and what they do

Last Week Recap

- C++

Game Architecture

Let's start with something you know, i.e. game architecture and draw a sequence diagram.

- Start
- Init
- Loop
- Destroy
- Exit

Game Architecture (Start)

- What is the entry point of a C++ program? What about Java?
- Take control from the environment
- Sanity check the system, e.g. graphics, audio, window

Game Architecture (Init)

- Load assets and data
- Show load screen or main menu

Game Architecture (Loop)

- Take input from player
- Update world based on input
- Render world
- Repeat . . .

Game Architecture (Destroy)

- Dispose of any accessed resources

Game Architecture (Exit)

- Give control back to the environment

Game Engine Architecture

Many interconnected subsystems, but if well-designed they form a powerful framework.

Image from Game Engine Architecture (get the (e)-book from the library!)

game_engine_arch

Game Engines

- Different engines have different views and workflows
- Architecture-wise, if we look generally, it is similar

Game Engines (Unity Project Structure)

```
Assets/
    Scenes/
    Prefabs/
    Scripts/
    Materials/
    ...
```

Unity

unity

Game Engines (Unreal Project Structure)

```
Content/
    Blueprints/
    Maps/
    Materials/
    Particles/
    Textures/
    ...
```

Unreal

ue

Game Engines (Godot Project Structure)

```
Game/
    Maps/
    Tiles/
    Objects/
    Scripts/
    Music/
    Sounds/
    Images/
    ...
```

Godot

godot

Game Engines (FXGL Project Structure)

```
assets/
    textures/
    music/
    sounds/
    scripts/
    level/
    ...
```

FXGL

fxgl

Game Engine Architecture

Overall, looks *very* similar to game architecture!

- Start (sanity check)
- Init (graphics, audio, background threads, asset manager, event system, etc.)
- Loop: (update engine timer)
- Destroy (cleanup)
- Exit

Activity

Let's explore existing open-source game engines: 1. Find a repo on GitHub with game engine source code. For example: Godot, CRYENGINE 2. Identify (in their code) their versions of the entry point (start / main)

Engine Subsystems

This is a brief overview, next weeks each focus on one subsystem in more detail.

Refer to the StudentCentral engine subsystem diagram (in Study Materials).

Game World Subsystem

- Manage (store, update) many game objects simultaneously
- Allow queries

Physics Subsystem

- Compute valid positions of objects
- Simulate (fake) physics
- Detect collisions

Graphics Subsystem

- Draws objects (entities + UI) to the screen
- Computes complex post-processing effects
- Graphics system stack (hardware -> ... -> high-level API)

Event Subsystem

- Communication between subsystems
- Manage and dispatch in-game events, expose them to users

Audio Subsystem

- 3D positional sound
- Ambient effects
- Ability to control properties of audio being played (volume, rate)

AI Subsystem

- Provides behaviour to game objects
- A set of pathfinding and search algorithms
- Works as a human replacement in single player mode

UI Subsystem

- Manage the UI view of the game
- Provide simple objects for manipulation
- Complex animations with interpolations
- Front-end validation

Activity

Let's explore existing open-source game engines: 1. Find a repo on GitHub with game engine source code. For example: FXGL, Godot 2. Identify (in their code) their versions of "Game World", "Main Loop", "Render" and "Physics Tick/Update".

Asset (Resource) Manager Subsystem

- (Pre)load and store assets in an efficient way
- Allow easy access to stored assets

Profiling / Debugging Subsystem

- Manage crashes gracefully
- Provide important data to developers (stacktrace, performance)
- Attempt to identify what went wrong (PS4 has a powerful system info dump on crash)

Networking Subsystem

- Updates, DLC, one-time events (e.g. holidays), notifications, friends list
- Multiplayer
- Cloud saves

Maths Subsystem

- Simplify domain code by reusing existing functions
- Handle combat (e.g. critical strike) / gameplay specifics (e.g. chance to drop loot on death)

Platform Subsystem

- Identify platform specifics (OS, capabilities)
- Map engine code to appropriate back-end (similar to SDL2)

Scripting Subsystem

- Allow easy engine extensions (user content)
- Reduces compilation times

Activity

Order and justify the following **8** steps of the main loop: updateAI, checkInput, notifyCollisions, renderGame, handleInput, clearRender, checkCollisions, renderUI.

Conclusion

- One architecture to rule them all, many actual implementations
- This is only the surface – there are many more subsystems. Feel free to explore!

Tutorial

On StudentCentral